



07278

PATENT TRADEMARK OFFICE

41363map.doc

10/05/01

EXPRESS MAIL CERTIFICATE

Date: 5/14/01 Label No. 706740327-US  
 I hereby certify that, on the date indicated above, this paper or  
 fee was deposited with the U.S. Postal Service & that it was  
 addressed for delivery to the Assistant Commissioner for  
 Patents, Washington, DC 20231 by "Express Mail Post Office  
 to Addressee" service.

D. Davis  
 Name (Print)

D. Davis  
 Signature

## EXTENDING THE RANGE OF COMPUTATIONAL FIELDS OF INTEGERS

### FIELD OF THE INVENTION

[0001] The present invention relates to apparatus operative to accelerate cryptographic co-processing peripherals and additionally but not exclusively to an accelerated processing apparatus for polynomial based and prime number field arithmetic, extending the range of computational fields of integers and width of serial input operands in modular arithmetic public key cryptographic coprocessors designed for elliptic curve and RSA type computations.

### BACKGROUND OF THE INVENTION

[0002] Security enhancements and performance accelerations for computational devices are described in Applicant's U.S. Patents 5,742,530, hereinafter "P1", 5,513,133, 5,448,639, 5,261,001; and 5,206,824 and published PCT patent application PCT/IL98/00148 (WO98/50851); and corresponding U.S. Patent application 09/050958, hereinafter "P2", Onyszchuk et al's U.S. Patent 4,745,568; Omura et al's U.S. Patent 4,5877,627, and applicant's U.S. Patent Application 09/480,102; the disclosures of which are hereby incorporated by reference. Applicant's U.S. Patent 5,206,824 shows an early apparatus operative to implement polynomial based multiplication and squaring, which cannot perform operations in the prime number field, and is not designed for interleaving in polynomial based computations. An additional analysis is made of an approach to use the extension field in polynomial based arithmetic in Paar, C., F. Fleischmann and P. Soria-Rodriguez, "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents", IEEE Transactions on Computers, vol. 48, No. 10, October 1999, henceforth "Paar". W. Wesley Peterson and E.J. Weldon Jr., in the second edition of "Error-Correcting Codes", published by the MIT Press, Cambridge, Mass., 1972, pages 174-179, demonstrated circuits for performing division in the polynomial based residual number system  $GF(2^q)$ , hereinafter, "Peterson". Peterson's circuit can only be used in a device where the multiplier is exactly the length of the modulus. Typically, that would demand a device that would be more than twice as long as present devices, and would not be cost effective for compact

implementations. It could not be used in interleaved implementations, and could not be useful where  $\ell$  is longer than 1, as he has not provided an anticipatory device for determining the  $Y_0$  of a multibit character.

[0003] Whereas, Knuth [D. Knuth, *The art of computer programming*, vol. 2: *Seminumerical algorithms*, Addison-Wesley, Reading Mass., 1981] page 407, implies that using an ordinary division process on a single  $\ell$  bit character in polynomial based division, we can assume a method to anticipate the next character in the quotient, this invention discloses a method for anticipating the next character of a quotient deterministically using a logic configuration.

#### SUMMARY OF THE INVENTION

[0004] It is an aim of the present invention to provide a microelectronic specialized arithmetic unit operative to perform large number computations in the polynomial based and prime integer based number fields, using the same anticipating methods for simultaneously performing interleaved modular multiplication and reduction on varied radix multipliers.

[0005] A further aim of the invention also relates to a compact microelectronic specialized arithmetic logic unit, for performing modular and normal (natural, non-negative field of integers) multiplication, division, addition, subtraction and exponentiation over very large integers. When referring to modular multiplication and squaring using both Montgomery methods and a reversed format method for simplified polynomial based multiplication and squaring, reference is made to the specific parts of the device as a superscalar modular arithmetic coprocessor, or SMAP, or MAP or SuperMAP™, also as relates to enhancements existing in the applicant's U.S. Patent pending 09/050,958 filed March 31, 1998 and a continuation in part, 09/480,102 filed on January 10, 2000.

[0006] Preferred embodiments of the invention described herein provide a modular computational operator for public key cryptographic applications on portable Smart Cards, typically identical in shape and size to the popular magnetic stripe credit and bank cards. Similar Smart Cards as per applicant's technology of US Patent 5,513,133 and 5,742,530, and applicants above mentioned pending applications are being used in the new generation of public key cryptographic devices for controlling access to

computers, databases, and critical installations; to regulate and secure data flow in commercial, military and domestic transactions; to decrypt scrambled pay television programs, etc. and as terminal devices for similar applications. Typically, these devices are also incorporated in computer and fax terminals, door locks, vending machines, etc.

**[0007]** The preferred architecture is of an apparatus operative to be integrated to a multiplicity of microcontroller and digital signal processing devices, and also to reduced instruction set computational designs while the apparatus operates in parallel with the host's processing unit.

**[0008]** This embodiment preferably uses only one multiplying device which inherently serves the function of two or three multiplying devices, basically similar to the architecture described in applicant's 5,513,133 and further enhanced in U.S. Patent application 09/050,958 and PCT application PCT/IL98/0048. Using present conventional microelectronic technologies, the apparatus of the present invention may be integrated with a controlling unit with memories onto a smart card microelectronic circuit.

**[0009]** The main difference between hardware implementations in the polynomial based field, and in the prime number field, is that polynomial based additions and subtractions are simple XOR logic operations, without carry signals propagating from LS to MS. Consequently, there is no interaction between adjacent cells in the hardware implementation, and subtraction and addition are identical procedures. The earliest public notice that the authors are aware of was a short lecture by Marco Bucci of the Fondazione Ugo Bordoni, at the Eurocrypt Conference Rump Session in Perugia, Italy, in 1994, though, even then, this configuration was well known to engineers practiced in the art.

**[0010]** Previous applicant's apparati, described in P1 and P2, were typically prepared to efficiently compute elliptic curve cryptographic protocols in the  $GF(p)$  field. For use in the  $GF(2^q)$  field, in this invention, we show that as there is no interaction between adjacent binary bits in the polynomial field, computations can be processed efficiently, simultaneously performing reduction and multiplication on a superscalar multiplication device without introducing Montgomery functions and Montgomery parasites. Multiplication in  $GF(2^q)$  is performed as the computation machine preferably starts from the most significant partial products. Reduction is performed by adding as

many moduli as are necessary to reset MS ones to zeroes. As there is no carry out, in these additions, results are automatically modularly reduced. In this invention polynomial computations are performed using the same architecture, wherein, in  $GF(2^q)$  the operands are fed in MS characters first, wherein all internal carry signals are forced to zero.  $GF(p)$  computations are preferably executed as in P1 and P2, wherein LS characters are processed first and MS characters last.

[0011] The architecture has been extended to allow for a potentially faster progression, in that serial multipliers are  $l$  bit wide characters, and at each clock, an  $l$  bit character is emitted from the carry save accumulator, the CSA. This has somewhat complicated the anticipation process ( $Y_0$ ), in that for single bit wide busses, an inversion of an odd number over a mod  $2^x$  base is also an odd number, and the least significant bit of the multiplicand ( $J_0$ ) was always a one. For both number fields, however the reduction process is identical, assuming the switch out of (suppression of) carries, if we remember that our only aim is to output a  $k$  character zero string, and we regard the  $Y_0$  function only as a zero forcing vector.

[0012] The present invention also seeks to provide an architecture for a digital device, which is a peripheral to a conventional digital processor, with computational, logical and architectural novel features relative to the processes described in US Patent 5,513,133.

[0013] A concurrent process and a hardware architecture are provided, to perform modular exponentiation without division preferably with the same number of operations as are typically performed with a classic multiplication/division device, wherein a classic device typically performs both a large scale multiplication and a division on each operation. A particular feature of a preferred embodiment of the present invention is the concurrency of larger scale anticipatory zero forcing functions, the extension of number fields, and the ability to integrate this type unit for safe communications.

[0014] The advantages realized by a preferred embodiment of this invention result from a synchronized sequence of serial processes. These processes are merged to simultaneously (in parallel) achieve three multiplication operations on  $n$  character operands, using one multiplexed  $k$  character serial/parallel multiplier in  $n$  effective clock cycles, where the left hand final  $k$  characters of the result reside in the output buffer of the multiplication device. This procedure accomplishes the equivalent of three

multiplication computations in both fields, as described by Montgomery, for the prime number field and the equivalent of two multiplications and a division process in  $GF(2^q)$ .

[0015] By synchronizing loading of operands into the SuperMAP and on the fly detecting values of operands, and on the fly preloading and simultaneous addition of next to be used operands, the apparatus is operative to execute computations in a deterministic fashion. All multiplication and exponentiation circuitry is preferably added which on the fly preloads, three first  $k$  character variables for a next iteration squaring sequence. A detection device is preferably provided where only two of the three operands are chosen as next iteration multiplicands, eliminating  $k$  effective clock cycle wait states. Conditional branches are replaced with local detection and compensation devices, thereby providing a basis for a simple control mechanism. The basic operations herein described are typically executed in deterministic time in  $GF(p)$  using a device described in US Patent 5,513,133 to Gressel et al or devices as by STMicroelectronics in Rousset, France, under the trade name ST19-CF58.

[0016] The apparatus of the present invention has particularly lean demands on external volatile memory for most operations, as operands are loaded into and stored in the device for the total length of the operation. The apparatus preferably exploits the CPU onto which it is appended, to execute simple loads and unloads, and sequencing of commands to the apparatus, whilst the MAP performs its large number computations. Large numbers presently being implemented on smart card applications range from 128 bit to 2048 bit natural applications. The exponentiation processing time is virtually independent of the CPU, which controls it. In practice, architectural changes are typically unnecessary when appending the apparatus to any CPU. The hardware device is self-contained, and is preferably appended to any CPU bus.

[0017] In general, the present invention also relates to arithmetic processing of large integers. These large numbers are typically in the natural field of (non-negative) integers or in the Galois field of prime numbers,  $GF(p)$ , composite prime moduli, and polynomial based numbers in  $GF(2^q)$ . More specifically, preferred embodiments of the present invention seek to provide devices that can implement modular arithmetic and exponentiation of large numbers. Such devices are suitable for performing the operations of Public Key Cryptographic authentication and encryption protocols, which, in the prime number field work over increasingly large operands and which cannot be

executed efficiently with present generation modular arithmetic coprocessors, and cannot be executed securely in software implementations. Preferably, the same general architecture is used in elliptic curve implementations, on integers, which are orders of magnitude smaller. Using the novel reverse mode method of multiplication, polynomial arithmetic is advantageous as generating zeroes does not encumber computations with the parasitic  $2^n$  factor.

**[0018]** The architecture offers a modular implementation of large operand integer arithmetic, while allowing for normal and smaller operand arithmetic enabled by widening the serial single character bus, i.e., use of a larger radix. Typically, this is useful for accelerating computations, for reducing silicon area for implementing the SuperMAP, and for generating a device of length compatible with popular Digital Signal Processors (DSP).

**[0019]** For modular multiplication in the prime and composite field of odd numbers,  $A$  and  $B$  are defined as the multiplicand and the multiplier, respectively, and  $N$  is defined as the modulus in modular arithmetic.  $N$ , is typically larger than  $A$  or  $B$ .  $N$  also denotes the composite register where the value of the modulus is stored.  $N$ , is, in some instances, typically smaller than  $A$ .  $A$ ,  $B$ , and  $N$  are typically  $n$  characters long, where characters are typically one to 8 bits long.  $k$  the number of  $l$  bit characters in the size of the group defined by the size (number of cells) of the multiplying device. Similarly, in polynomial based  $GF(2^q)$  computations, the modulus,  $N$ , is  $n$  bits long wherein the MS bit is a one (a monic), and the  $A$ ,  $S$  and  $B$  operands are also, when properly reduced,  $n$  bits long. If a result of a  $GF(2^q)$ , computation is monic it is preferably “reduced” to a value with an MS zero, by XORing said result value with the modulus. In a preferred embodiment, as the first significant bit of a  $GF(2^q)$  is formed in the reverse mode, the MAP can sense if the bit is a one, and perform the preferred reduction.

**[0020]** In the prime field,  $\equiv$ , or in some instances  $=$ , is used to denote congruence of modular numbers, for example  $16 \equiv 2 \pmod{7}$ . 16 is termed “congruent” to 2 modulo 7 as 2 is the remainder when 16 is divided by 7. When  $Y \pmod{N} \equiv X \pmod{N}$ ; both  $Y$  and  $X$  may be larger than  $N$ ; however, for positive  $X$  and  $Y$ , the remainders are identical. Note also that the congruence of a negative integer  $Y$ , is  $Y + u \cdot N$ , where  $N$  is the modulus, and

if the congruence of  $Y$  is to be less than  $N$ ,  $u$  is the smallest integer which gives a positive result.

**[0021]** In  $GF(2^q)$  congruence is much simpler, as addition and subtraction are identical, and normal computations typically do not leave a substantial overflow. For  $N = 1101$  and  $A = 1001$ , as the left hand MS bit of  $A$  is 1, we must reduce ("subtract")  $N$  from  $A$  by using modulo 2 arithmetic,  $A \text{ XOR } N = 1001 \text{ XOR } 1101 = 0100$ .

**[0022]** The Yen symbol, ¥, is used to denote congruence in a limited sense, especially useful in  $GF(p)$ . During the processes described herein, a value is often either the desired value, or equal to the desired value plus the modulus. For example  $X \text{ ¥ } 2 \text{ mod } 7$ .  $X$  can be equal to 2 or 9.  $X$  is defined to have limited congruence to 2 mod 7. When the Yen symbol is used as a superscript, as in  $B^¥$ , then  $0 \leq B^¥ < 2N$ , or stated differently,  $B^¥$  is either equal to the smallest positive  $B$  which is congruent to  $B^¥$ , or is equal to the smallest positive congruent  $B$  plus  $N$ , the modulus. Other symbols, specific to this invention appear later in this summary.

**[0023]** When  $X = A \text{ mod } N$ ,  $X$  is defined as the remainder of  $A$  divided by  $N$ ; e.g.,  $3 = 45 \text{ mod } 7$ , and much simpler in  $GF(2^q) - 1111 \text{ mod } 1001 \equiv 0110$ .

**[0024]** In number theory, the modular multiplicative inverse of  $X$  is written as  $X^{-1}$ , which is defined by  $X \cdot X^{-1} \text{ mod } N = 1$ . If  $X = 3$ , and  $N = 13$ , then  $X^{-1} = 9$ , i.e., the remainder of  $3 \cdot 9$  divided by 13 is 1 in  $GF(p)$ .

**[0025]** For both number fields, we typically choose to compute the multiplicative inverse of  $A$  using the exponential function, e.g.,  $A^{-1} \text{ mod } q \equiv A^{q-2} \text{ mod } q$ .

**[0026]** The acronyms MS and LS are used to signify "most significant" and "least significant", respectively, when referencing bits, characters, and full operand values, as is conventional in digital nomenclature, but in the reversed mode polynomial base, operands are loaded MS data first and LS last, wherein the bit order of the data word is reversed when loaded.

**[0027]** Throughout this specification  $N$  designates both the value  $N$ , and the name of the shift register which stores  $N$ . An asterisk superscript on a value, denotes that the value, as stands, is potentially incomplete or subject to change.  $A$  is the value of the number which is to be exponentiated, and  $n$  is the bit length of the  $N$  operand. After initialization when  $A$  is "Montgomery  $P$  field normalized" to  $A^*$  ( $A^* = 2^n A$  - explained in P1)  $A^*$  and  $N$  are typically constant values throughout the intermediate step in the

exponentiation. In  $GF(2^q)$  computations where computations might be performed with the normal unreversed positioning of bits we would be bound by this same protocol. However, using the reversed format, our computations generate most significant zeroes, which are disregarded, and do not represent a multiplication shift, as there is no carry out.

**[0028]** During the first iteration, after initialization of an exponentiation,  $B$  is equal to  $A^*$ .  $B$  is also the name of the register wherein the accumulated value that finally equals the desired result of exponentiation resides.  $S$  or  $S^*$  designates a temporary value; and  $S$  designates the register or registers in which all but the single MS bit of a number in  $GF(p)$   $S$  is stored. ( $S^*$  concatenated with this MS bit is identical to  $S$ .)  $S(i-1)$  denotes the value of  $S$  at the outset of the  $i$ 'th iteration. In these polynomial computations there is no need to perform modular reduction on  $S$ .

**[0029]** Typically, Montgomery multiplication of  $X$  and  $Y$  in the prime number field is actually an execution of  $(X \cdot Y \cdot 2^{-n}) \bmod N$ , where  $n$  is typically the number of characters in a modulus. This is written,  $\mathcal{P}(A \cdot B)_N$ , and denotes MM or multiplication in the  $P$  field. In the context of Montgomery mathematics, we refer to multiplication and squaring in the  $P$  field and in the polynomial based field as multiplication and squaring operations.

**[0030]** We will redefine this innovative extension of a Montgomery type arithmetic in the  $GF(2^q)$  to mean a reversed format data order, wherein MS zero forcing does not change congruence, or initiate a burdensome parasitic factor. We may thus introduce a new set of symbols to accommodate the arithmetic extension, and to enable an architecture with wider serial multiplier buses. Such a more than one bit serial multiplier stream is preferable for enabling a natural integer superscalar multiplier. Such multiplier device may accept 32 bit multiplicands and 4 bit multipliers, into an apparatus that can perform modular arithmetic multiplications and reductions simultaneously.

### **[0031] Symbols in the Serial/Parallel Super Scalar Modular Multiplier Enhancement**

$\ell$  Number of bits in a character (digit).

$r$  Radix of multiplier character,  $r = 2^\ell$ .



- $n$  Size of operands (multiplier, multiplicand and modulus) in characters. In the demonstration of the computations in the  $\text{GF}(p)$  field of Montgomery arithmetic,  $l$  is equal to one, and  $n$  is the bit length of the modulus operand.
- $k$  Length of serial-parallel multiplier in characters.
- $m$  Number of interleaved slices (segments) of multiplicand:  $m = n/k$ .
- $S_i$  Partial product result of  $i$  'th MM iteration;  $0 \leq i \leq m - 1$ ;  $S_0 = 0$ .
- $S_{i0}$  Right hand character of the  $i$  'th iteration result, after **disregarding** the first  $k$  character right hand zeroes of  $Z$ .
- $\underline{S}_i$  The left hand  $n - k$  characters of the  $i$  'th result.
- $S_{ij}$   $j$  'th character of  $S_i$ .
- $A$  Parallel multiplicand consists of  $m \cdot k$  characters.
- $A_i$  The  $i$  'th  $k$  character slice of  $A$ , (and/or register storing  $A_i$ )
- $A_{it}$  The  $t$  'th character of  $A_i$ .
- $B$  A serial multiplier (and/or register store of  $B$ ).
- $B_0$  First right hand  $k$  characters of  $B$ .
- $\underline{B}$  Last left hand  $(n - k)$  characters of  $B$ .
- $B_{0j}$   $j$  'th character of  $B_0$ .
- $\underline{B}_j$   $j$  'th character of  $\underline{B}$ .
- $N$  Modulus operand. (and/or register storing said multiplier).
- $N_0$  The Right Hand  $k$  characters of  $N$ .  
{the LS characters in  $\text{GF}(p)$ ; MS characters in  $\text{GF}(2^q)$  }
- $\underline{N}$   $(n - k)$  Left Hand characters of  $N$ .  
{ MS characters in  $\text{GF}(p)$ ; LS characters in  $\text{GF}(2^q)$  }
- $N_{0j}$   $j$  'th character of  $N_0$ .
- $\underline{N}_j$   $j$  'th character of  $\underline{N}$ .
- $Y_0$  Zero forcing variable required for both Montgomery multiplication and reduction in  $\text{GF}(p)$ .  $Y_0$  is  $k$  characters long.
- $Y_{0j}$   $j$  'th character of  $Y_0$ .
- $R$  A summation of the value residing in the Carry Save Accumulator, (includes unresolved internal carry ins) and the carry out bit from the final serial summator, 460.

$J_{00}$  Zero forcing character function of the modulus,  $N$ , for "on the fly" finite field multiplication and reduction. For  $\ell = 1$ ,  $J_{00}$  is always equal to 1.

$Carry_j$   $j$ 'th internal carry character of radix  $r$  serial-parallel multiplier.

$Carry_a$  radix  $r$  carry of output serial adder for  $GF(p)$  computations.

$Sum_j$   $j$ 'th internal sum character of radix  $r$  serial-parallel multiplier.

LS Least Significant.

MS Most Significant.

$\parallel$  Concatenation, e.g.  $A = 110$ ,  $B = 1101$  ;  $A \parallel B = 1101101$ .

Right Hand A Least Significant portion of all  $GF(p)$  computation data blocks and an MS portion of the reversed  $GF(2^q)$  format.

Left Hand A Most Significant portion of all  $GF(p)$  computation data blocks and an LS portion of the reversed  $GF(2^q)$  format.

$GF(p)$  Galois Field, strictly speaking finite fields over prime numbers where we also use composite integers (product of two very large prime numbers) that allow for addition, subtraction, multiplication and pseudo-division.

$GF(2^q)$  Galois Fields finite fields using modulo 2 arithmetic.

$\oplus$  An operator or device which may be externally switched to add or subtract integers with or without carries, as befits the specific number system.

$\otimes$  An operator or device which may be switched to either execute multiplication over  $GF(p)$  or multiplication over  $GF(2^q)$ .

$\mathcal{S}$  The number field switch, where if:

$\mathcal{S} = 1$ , the switch is operative to enable all carry in/outputs for  $GF(p)$  computations;

$\mathcal{S} = 0$ , the switch is operative to disable all carry in/outputs for  $GF(2^q)$  computations.

SuperMAP Any one of a member of the proprietary Superscalar Modular Arithmetic Processor family that is the subject of this invention. The SuperMAP trademark is registered in Europe and is pending in the United States.

**[0032]** According to a first aspect of the present invention, there is therefore provided a microelectronic apparatus for performing  $\otimes$  multiplication and squaring in both polynomial based  $GF(2^q)$  and  $GF(p)$  field arithmetic, squaring and reduction using a serial fed radix  $2^\ell$  multiplier,  $B$ , with  $k$  character multiplicand segments,  $A_i$ , and a  $k$  character  $\oplus$  accumulator wherein reduction to a limited congruence is performed "on

the fly", in a systolic manner, with  $A_i$ , a multiplicand, times  $B$ , a multiplier, over a modulus,  $N$ , and a result being at most  $2n + 1$  characters long, including the  $k$  first emitting disregarded zero characters, which are not saved, where  $k$  characters have no less bits than the modulus, wherein said operations are carried out in two phases, the apparatus comprising;

[0033] a first ( $B$ ), and second ( $N$ ) main memory register means, each register operative to hold at least  $n$  bit long operands, respectively operative to store a multiplier value designated  $B$ , and a modulus, denoted  $N$ , wherein the modulus is smaller than  $2^n$ ;

[0034] a digital logic sensing detector,  $Y_0$ , operative to anticipate "on the fly" when a modulus value is to be  $\oplus$  added to the value in the  $\oplus$  adder accumulator device such that all first  $k$  characters emitting from the device are forced to zero;

[0035] a modular multiplying device for at least  $k$  character input multiplicands, with only one, at least  $k$  characters long  $\oplus$  adder,  $\oplus$  summation device operative to accept  $k$  character multiplicands, the  $\otimes$  multiplication device operative to switch into the  $\oplus$  accumulator device, in turn, multiplicand values, and in turn to receive multiplier values from a  $B$  register, and an "on the fly" simultaneously generated anticipated value as a multiplier which is operative to force  $k$  first emitting zero output characters in the first phase, wherein at each effective machine cycle at least one designated multiplicand is  $\oplus$  added into the  $\oplus$  accumulation device;

the multiplicand values to be switched in turn into the  $\oplus$  accumulation device consisting of one or two of the following three multiplicands, the first multiplicand being an all-zero string value, a second value, being the multiplicand  $A_i$ , and a third value, the  $N_0$  segment of the modulus;

an anticipator to anticipate the  $\angle$ bit  $k$  character serial input  $Y_0$  multiplier values;

the apparatus being operable to input in turn multiplier values into the multiplying device said values being first the  $B$  operand in the first phase, and concurrently, the second multiplier value consisting of the  $Y_0$ , "on the fly" anticipated  $k$  character string, to force first emitting zeroes in the output;

the apparatus further comprising an accumulation device,  $\oplus$ , operative to output values simultaneously as multiplicands are  $\oplus$  into the  $\oplus$  accumulation device;

an output transfer mechanism, in the second phase operative to output a final modular  $\otimes$  multiplication result from the  $\oplus$  accumulation device.

[0036] According to a preferred embodiment  $\oplus$  summations into the  $\oplus$  accumulation device are activated by each new serially loaded higher order multiplier characters.

[0037] Preferably, the multiplier characters are operative to cause no  $\oplus$  summation into the  $\oplus$  accumulation device if both the input  $B$  character and the corresponding input  $Y_0$  character are zeroes;

are operative to  $\oplus$  add in only the  $A_i$  multiplicand if the input  $B$  character is a one and the corresponding  $Y_0$  character is a zero;

are operative to  $\oplus$  add in only the  $N$ , modulus, if the  $B$  character is a zero, and the corresponding  $Y_0$  character is a one; and

are operative to  $\oplus$  add in the  $\oplus$  summation of the modulus,  $N$ , with the multiplicand  $A_i$  if both the  $B$  input character and the corresponding  $Y_0$  character are ones.

[0038] Preferably, the apparatus is operative to preload multiplicand values  $A_i$  and  $N$ , into two designated preload buffers, and to  $\oplus$  summate these values into a third multiplicand preload buffer, obviating the necessity of  $\oplus$  adding in each multiplicand value separately.

[0039] Preferably, the multiplier character values are arranged for input in serial single character form, and wherein the  $Y_0$  detect device is operative to anticipate only one character in a clocked turn.

[0040] In a preferred embodiment wherein the  $\oplus$  accumulation device performs modulo 2 computations, XOR addition/subtraction, wherein all carry bits in addition and subtraction components are disregarded, thereby precluding provisions for overflow and further limiting convergence in computations.

[0041] Preferably, carry inputs are disabled to zero, denoted,  $\mathcal{S}=0$ , typically operative to perform polynomial based multiplication.

[0042] Preferably the apparatus is operative to provide non-carry arithmetic by omitting carry circuitry, such that an  $\mathcal{S}$  equal to zero acting on an element in a circuit

equation computing in  $GF(2^q)$ ,  $\mathcal{S}$  designates omitted circuitry and all adders and subtractors, designated  $\oplus$  to XOR, modulo 2 addition/subtraction elements.

[0043] A preferred embodiment is adapted such that the first  $k$  character segments emitted from the operational unit are zeroes, being controlled by the following four quantities in anticipating the next in turn  $Y_0$  character:

- i. the  $\ell$  bit  $S_{out}$  bits of the result of the  $\ell$  bit by  $\ell$  bit mod  $2^\ell \otimes$  multiplication of the right-hand character of the  $A_i$  register times the  $B_d$  character of the  $B$  Stream,  $A_0 \otimes B_d \bmod 2^\ell$ ;
- ii. the first emitting carry out character from the  $\oplus$  accumulation device,  $\mathcal{S}(CO_0)$ ;
- iii. the  $\ell$  bit  $S_{out}$  character from the second from the right character emitting cell of the  $\oplus$  accumulation device,  $SO_1$ ;
- iv. the  $\ell$  bit  $J_0$  value, which is the negative multiplicative inverse of the right-hand character in the  $N_0$  modulus multiplicand register,

wherein values,  $A_0 \otimes B_d \bmod 2^\ell$ ,  $\mathcal{S}(CO_0)$ , and  $SO_1$  are  $\oplus$  added character to character together and "on the fly" multiplied by the  $J_0$  character to output a valid  $Y_0$  zero-forcing anticipatory character to force an  $\ell$  bit egressing string of zeroes.

[0044] The apparatus is preferably operable to perform multiplication on polynomial based operands performed in a reverse mode, multiplying from right hand MS characters to left hand LS characters, operative to perform modular reduced  $\otimes$  multiplication without Montgomery type parasitic functions.

[0045] Preferably, the apparatus further comprises preload buffers which are serially fed and where multiplicand values are preloaded into the preload buffers on the fly from one or more memory devices.

[0046] The apparatus is preferably operative to  $\oplus$  summate into a multiplication stream a previous value, emitting from an additional  $n$  bit  $S$  register, via an  $\ell$  bit  $\oplus$  adder circuit such that first emitting output characters are zeroes when the  $Y_0$  detector is operative to detect the necessity of  $\oplus$  adding moduli to the  $\oplus$  summation in the  $\oplus$  accumulation device, wherein the  $Y_0$  detector operates to detect, utilizing the next in

turn  $\oplus$  added characters  $A_0 \otimes B_d \bmod 2^\ell$ ,  $\mathcal{S}(CO_0)$ ,  $SO_1, S_d$  and  $\mathcal{S}(CO_z)$ , the composite of  $\oplus$  added characters to be finite field  $\otimes$  multiplied on the fly by the  $\ell$  bit  $J_0$  value, where  $\oplus$  defines the addition and  $\otimes$  defines the multiplication as befits the finite field used in the process.

[0047] Preferably, for  $\ell = 1$ ,  $J_0$  is implicitly 1, and the  $J_0 \otimes$  multiplication is implicit, without additional hardware.

[0048] Preferably, a comparator is operative to sense a finite field output from the  $\otimes$  modular multiplication device, whilst working in  $GF(p)$ , where the first right hand emitting  $k$  zero characters are disregarded, where the output is larger than the modulus,  $N$ , thereby operative to control a modular reduction whence said value is output from the memory register to which the output stream from the multiplier device is destined, and thereby precluding allotting a second memory storage device for the smaller product values.

[0049] Preferably, for  $\otimes$  modular multiplication in the  $GF(2^q)$ , the apparatus is operative to multiply without an externally precomputed more than  $\ell$  bit zero-forcing factor.

[0050] A preferred embodiment is operative to compute a  $J_0$  constant by resetting either the  $A$  operand value or the  $B$  operand value to zero and setting the partial result value,  $S_0$ , to 1.

[0051] According to a second aspect of the present invention there is provided a microelectronic apparatus for performing interleaved finite field modular multiplication of integers  $A$  and  $B$ , so as to generate an output stream of  $A$  times  $B$  modulus  $N$ , wherein a number of characters in a modulus operand register,  $n$ , is larger than a segment length of  $k$  characters, wherein the  $\otimes$  multiplication process is performed in a plurality of interleaved iterations, wherein at each interleaved iteration with operands input into a  $\otimes$  multiplying device, consisting of  $N$ , the modulus,  $B$ , a multiplier, a previously computed partial result,  $S$ , and a  $k$  character string segment of  $A$ , a multiplicand, the segments progressing from the  $A_0$  string segment to the  $A_{m-1}$  string segment, wherein each iterative result is  $\oplus$  summated into a next in turn  $S$ , temporary result, in turn, wherein first emitting characters of iterative results are zeroes, the apparatus comprising:

first ( $B$ ), second ( $S$ ) and third ( $N$ ) main memory registers, each register respectively operative to store a multiplier value, a partial result value and a modulus;

a modular multiplying device operative to  $\oplus$  summate into the  $\oplus$  accumulation device, in turn one or two of a plurality of multiplicand values, during each one of a plurality of phases of the iterative  $\otimes$  multiplication process, and in turn to receive as multipliers, in turn, inputs from:

said  $B$  register,

an "on the fly" anticipating value,  $Y_0$ , being usable as a multiplier to force first emitting right-hand zero output characters in each iteration, and

said  $N$ , register;

the multiplicand parallel registers operative at least to receive in turn, values from the  $A$ ,  $B$ , and  $N$  register sources, and in turn, also a multiplicand zero forcing  $Y_0$ , value;

the apparatus further utilizing the  $Y_0$  detect device operative to generate a binary string operative to be a multiplier during the first phase and operative to be a multiplicand in the second phase;

the apparatus being operable to obtain multiplicand values suitable for switching into the  $\oplus$  accumulation device for the first phase consisting of a first zero value, a second value,  $A_i$ , which is a  $k$  character string segment of a multiplicand,  $A$ , and a third value  $N_0$ , being the first emitting  $k$  characters of the modulus,  $N$ ;

the apparatus further being operable to utilize a temporary result value,  $S$ , resulting from a previous iteration, operative to be  $\oplus$  summated with the value emanating from the  $\oplus$  accumulation device, to generate a partial result for the next-in-turn iteration;

the apparatus further being operable to utilize multiplicand values to be input, in turn, into the  $\oplus$  accumulation device for a second multiplication phase being, comprising firstly a first zero value, a second  $A_i$  operand, remaining in place from the first phase, and thirdly a  $Y_0$  value having been anticipated in the first phase;

multiplier values input into the multiplying device in the first phase being firstly an emitting string,  $B_0$ , said multiplication device being operable to multiply said string segment concurrently  $\otimes$  multiplying with a second  $\otimes$  multiplier value consisting

of the anticipated  $Y_0$  string which is simultaneously loaded character by character as it is generated into a preload multiplicand buffer for the second phase;

the two multiplier values operative to be input into the apparatus during a second phase being left hand  $n - k$  character values from the  $B$  operand, designated  $\underline{B}$ , and the left hand  $n - k$  characters of the  $N$  modulus, designated  $\underline{N}$ , respectively; and

wherein said apparatus further comprises a multiplying flush out device operative in a last phase to transfer the left hand segment of a result value remaining in the  $\oplus$  accumulation device into a result register.

**[0052]** Preferably, the apparatus is operable to perform  $\otimes$  multiplication on polynomial based operands is performed in a reverse mode, multiplying from MS characters to LS characters, operative to perform modular reduction without the Montgomery type parasitic functions, as in applicant's patent US 5,742,530.

**[0053]** According to a third aspect of the present invention there is provided apparatus operative to anticipate a  $Y_0$  value using first emitting values of the multiplicand, and present inputs of the  $B$  multiplier, carry out values from the  $\oplus$  accumulation device,  $\oplus$  summation values from the  $\oplus$  accumulation device, the present values from the previously computed partial result, and carry out values from the  $\oplus$  adder which  $\oplus$  summates the result from the  $\oplus$  accumulation device with the previous partial result.

**[0054]** Preferably, the apparatus is adapted to insure that  $k$  first emitting values from the device are zero characters said adaptation comprising anticipation of a next in turn  $Y_0$  character using the following quantities:

- i. the  $\ell$  bit  $S_{out}$  bits of the result of the  $\ell$  bit by  $\ell$  bit mod  $2^\ell \otimes$  multiplication of the right-hand character of the  $A_i$  register times the  $B_d$  character of the  $B$  Stream,  $A_0 \cdot B_d \text{ mod } 2^\ell$ ;
- ii. the first emitting carry out character from the  $\oplus$  accumulation device,  $\mathcal{S}(CO_0)$ ;
- iii. the  $\ell$  bit  $S_{out}$  character from the second from the right-hand character emitting cell of the  $\oplus$  accumulation device,  $SO_1$ ;
- iv. the next in turn character value from the  $S$  stream,  $S_d$ ;
- v. the  $\ell$  bit carry out character from the  $Z$  output full adder,  $\mathcal{S}(CO_z)$ ;



vi. the  $\angle$ bit  $J_0$  value, which is the negative multiplicative inverse of the right-hand character in the  $N_0$  modulus multiplicand register;

wherein values,  $A_0 \cdot B_d \bmod 2^l$ ,  $\mathcal{S}(CO_0)$ ,  $SO_1$ ,  $S_d$  are  $\oplus$  added character to character together and "on the fly"  $\otimes$  multiplied by the  $J_0$  character to output a valid  $Y_0$  zero-forcing anticipatory character.

[0055] In a further embodiment there is also provided at least one sensor operative to compare the output result to  $N$ , the modulus, the mechanism operative to actuate a second subtractor on the output of the result register, thereby to output a modular reduced value which is of limited congruence to the output result value precluding the necessity to allot a second memory storage for a smaller result.

[0056] In a yet further embodiment, a value which is a  $\oplus$  summation of two multiplicands is loaded into a preload character buffer with at least a  $k$  characters memory means register concurrently whilst one of the first values is loaded into another preload buffer.

[0057] According to a fourth aspect of the present invention there is provided apparatus with one  $\oplus$  accumulation device, and an anticipating zero forcing mechanism operative to perform a series of interleaved  $\otimes$  modular multiplications and squarings and being adapted to perform concurrently the equivalent of three natural integer multiplication operations, such that a result is an exponentiation.

[0058] In an embodiment, next-in-turn used multiplicands are preloaded into a preload register buffer on the fly.

[0059] In a further embodiment, apparatus buffers and registers are operative to be loaded with values from external memory sources and said buffers and registers are operative to be unloaded into the external memory source during computations, such that the maximum size of the operands is dependent on available memory means.

[0060] In a yet further embodiment there is also provided memory register means, said memory means are typically serial single character in/ serial single character out, parallel at least  $k$  characters in/parallel at least  $k$  characters out, serial single character in/parallel at least  $k$  characters out, and parallel  $k$  characters in/serial single character out.

[0061] Preferably the apparatus is operable to provide, during a final phase of a multiplication type iteration, the multiplier inputs are zero characters which are operative to flush out the left hand segment of the carry save  $\oplus$  accumulator memory.

[0062] Preferably, the apparatus is operable to preload next in turn multiplicands into preload memory buffers on the fly, prior to their being required in an iteration.

[0063] Preferably, the apparatus is operable to preload multiplicand values into preload buffers on the fly from central storage memory means.

[0064] The same device is operable to compute the  $k$  character Montgomery constant  $J_0$ , related to the right hand  $k$  character segment of a modulus preferably by resetting both  $A$  and  $B$  to zero and setting  $S_0 = 1$ , whilst subsequently performing a  $k$  bit multiplication. The result will reside in the  $Y_0$  register.

### **Modular Multiplication Sequences using Montgomery type Arithmetic**

[0065] The  $k$  character carry save adder, the CSA, is the basis for the serial/parallel superscalar modular multiplication in both the polynomial field and in the prime number field. Polynomial  $GF(2^q)$  based computations are executed preferably with all carry mechanisms switched off.

[0066] The Serial - Parallel Super Scalar Montgomery Multiplier computes Montgomery modular product in three phases, wherein in one preferred embodiment, the last phase may be a single clock dump of the whole left hand  $k$  character segment with a carry of the CSA (MS for normal multiplications, LS for reverse mode polynomial computations), and in a more compact embodiment, the last phase may be a  $k$  effective clock serial flush out of the contents of the CSA.

[0067] In previous P2 disclosures, the  $Y_0$  factor was computed bit by bit, consequently, only the right hand bit of  $J_0$  was consequential, and by definition a one, and a function of the right hand bit of the modulus. In this enhancement device, the device is character serial, and an  $l$  bit character  $Y_0$  is generated at each clock cycle. As in previous P1 disclosures,  $Y_0$  is the first phase zero forcing function which adds in the value of modulus a necessary number of times into the accumulated result, so that the relevant answer is congruent and never longer than  $mk+1$  characters, and the first right hand emitted characters were all zeroes.  $X + QN \equiv X$ , as  $QN \equiv 0 \pmod{N}$ .

## Modular Multiplication Sequences

[0068] Prior to initiating computation, we assume that there are no previous temporary or random values in the device; that the operands  $N$ ,  $B$ , and at least the first segment value of  $A$  are available in the registers of the device.  $S_0 = 0$ ; the partial product at initiation is typically zero. Typically, modular arithmetic is executed on operands consisting of two or more  $k$  character segments, typically in three distinct phases. For a normal full multiplication, where there are  $m$  segments of the modulus, there are typically  $m$  superscalar multiplication interleaved iterations whence each segment of the multiplicand is multiplied by the total multiplier, typically,  $B$ .

[0069] The process of the first phase, on the ( $i = 0$ 'th segment) of each interleaved superscalar multiplication is the generic superscalar multiplication accumulation interaction:

$$S_i \oplus A_i \cdot B_0 \oplus Y_0 \cdot N_0$$

[0070] ( $B_0$  and  $Y_0$  are serially character by character fed from the first segment of the operand into the multiplier,  $A_i$  and  $N_0$  are parallel single slice operands,  $S_i$  is the partial product from a previous iteration/computation.  $S_i = 0$  on the 0'th first iteration.)

[0071] The first phase process implements a  $\oplus$  summation of the two superscalar products with the right hand segment of the previous result. A  $k$  character string of zeroes has emitted from the multiplying device, and is disregarded; a partial first segment result resides in the device buffer, which is summated into the second phase result.

[0072] The first phase result consists typically of  $R$ , the contents of the CSA concatenated with the serial outputted right hand segment of all zeroes. (In GF(p) computations there is an additional LS carry-in bit to  $R$ .)

[0073] The process of the second phase is the generic superscalar multiplication accumulation interaction:

$$R \oplus S_i \oplus A_i \cdot \underline{B} \oplus Y_0 \cdot \underline{N}$$

[0074] (Remember, an underlined variable, e.g.,  $\underline{B}$ , is the remaining left hand value of an operand. It is typically one or more segments, i.e.,  $m - 1$  segments.  $\underline{B}$  and  $\underline{N}$  are serially fed character by character into the multiplier, and both  $A_i$ , remaining from the first phase, and  $Y_0$ , which was a multiplier in the first phase and was loaded into the

machine in the first phase to be a multiplicand in subsequent iterations, are parallel operands).

[0075] At the end of the second phase, typically consisting of  $m-1$  iterations, the left hand segment of  $S_i$  remains in the CSA - ready to be transferred – and the right hand slice ( $k$  character segment) has emanated from the device, typically into an  $S$  register. Note that multiplication in the prime number field has been performed in a conventional carry save summation method. Multiplication in the  $GF(2^q)$  reversed format mode has progressed from most to least significant characters. The  $Y_0$  function has anticipated when a modulus value must be "added" into the accumulator. Except for the disabled carry bits in the device, the mechanical process is typically identical for the two number systems.

[0076] We disclose the method used for deriving the  $Y_{0j}$  characters of the zero-forcing vector in finite fields.

[0077] Compute:  $J_{0j} \equiv -N_{00}^{-1} \bmod 2^l$ .

[0078] All natural integers which are relatively prime to a modulus have multiplicative inverses in both number fields.  $N_{00}$  is odd and, therefore, has no factor of 2. All factors of  $\bmod 2^l$  are 2, so that any number with a least significant 1, and a modulus whose only factor is 2, are relatively prime, and  $J_{0j}$  always exists. Formally, for odd  $N_{00}$  and  $2^l$ ,  $\gcd(N_{00}, 2^l) = 1$ .

[0079] This single character of the function can be hardware implemented with random logic, with simple circuitry, or with a simple look up table. There are only  $2^{k-1}$  different values that must be derived in a look up table. In the reverse mode format, the polynomial modulus must be right justified, nominally odd. In typical exponential functions, both number fields, the right hand bit of the modulus bit is a one, nominally odd, and the multiplicative inverse of an odd number  $\bmod 2^k$  must always be odd.

[0080] If  $l \neq 1$ , the  $J_{00}$  multiplier is explicitly equal to 1, and need not be computed. The result of a character output forced by the  $Y_0$  function during the first phase is always 0, consequently the  $j$ 'th character output,  $Z_{ij}$ , of the SuperMAP is:

$$0 = (2^l R \oplus S_{ij} \oplus A_{i0} \cdot B_{0j} \oplus Y_{0j} \cdot N_{00}) \bmod 2^l = Z_{ij}; \text{ therefore,}$$

$$(R \oplus S_{ij} \oplus A_{i0} \cdot B_{0j}) \equiv -Y_{0j} \cdot N_{00}; \text{ and}$$

$$Y_{0j} \equiv -N_{00}^{-1} (R \oplus S_{ij} \oplus A_{i0} \cdot B_{0j}) \bmod r.$$

**[0081]** From the above equation, we learn that  $J_{00}$  is preferably the negative value of the modular multiplicative inverse of the right hand  $k$  character of the modulus for both number systems; noting that in modulo 2 arithmetic, positive and negative values are the same.

**[0082]**  $R$  is the summation of the value remaining in the CSA summated to the carry out bit from the final serial adder, 460, in Fig. 2.  $S_{ij}$  is the  $j$ 'th bit of the partial product at the  $i$ 'th iteration.  $A_{i0}$  is the right hand character (LS in  $GF(p)$ ) of the  $i$ 'th slice of  $A$ .  $B_{0j}$  is the  $j$ 'th character of  $B$ .  $B_0$  is a constant (multiplier) during all iterations of a Montgomery multiplication.  $Y_0$  is a  $k$  character vector generated at each ( $i$ 'th) iteration.  $Y_{0j}$  is  $j$ 'th character generated at the  $j$ 'th clock of the first phase of an iteration.  $N$ , is an  $m$  sliced modulus.  $N_0$  is the right hand slice of the modulus.  $N_{00}$  is the right hand character of  $N_0$ .

**[0083] Formalizing the Super scalar modular multiplication method for both fields:**

$$S_0 = 0;$$

For  $i = 0$  to  $m - 1$  (interleave iterations)

First phase: (of each interleave)

$$R = 0$$

For  $j = 0$  to  $k - 1$  (each character of first phase)

$$Y_{0j} = (J_{00} (R \oplus S_{0j} \oplus A_{i0} \otimes B_{0j})) \bmod 2^l$$

$$Z_{ij} = (R \oplus S_{ij} \oplus A_{i0} \otimes B_{0j} \oplus Y_{0j} \otimes N_0) \bmod 2^l; \text{ and}$$

$$R = [(2^l R \oplus S_{ij} \oplus A_{i0} \otimes B_{0j} \oplus Y_{0j} \otimes N_0)] / 2^l$$

After  $k$  effective clock cycles, the first segment of the  $Z$  stream was all zeroes, and was disregarded; the relevant  $Y_0$ ,  $k$  character vector, is now prepared to be a multiplicand in the next phase, and the summated  $R$  value will be used in the next phase.

Second phase:

For  $j = k$  to  $n - 1$

$$\underline{Z}_{ij} = (R \oplus S_{ij} \oplus A_{i0} \otimes B_{0j} \oplus Y_{0j} \otimes N_{0j}) \bmod 2^l;$$

$$R = [2^l R \oplus S_{ij} \oplus A_{i0} \otimes B_{0j} \oplus Y_{0j} \otimes N_{0j}] / 2^l$$

Implementation of the above algorithm with a character based serial-parallel multiplier is a simple extension of the above protocol:

(*Quotient*(  $x, y$  ) is the integer division function without remainder. For example, if  $x = 10101_b$ , and  $y = 10000_b$ , then *Quotient*(  $x, y$  ) = 1 ).

$$S_0 = 0$$

For  $i = 0$  to  $m - 1$  (Interleaved loop)

First phase:

For  $j = 0$  to  $k - 1$

$$Y_{0j} =$$

$$(J_{00} \cdot S_{i0} \oplus A_{i0} \otimes B_{0j} \oplus \mathcal{S} \cdot Carry_0 \oplus Sum_1 \oplus Quotient(S_{i0} \oplus Sum_0, r)) \bmod r$$

For  $t = 0$  to  $k - 1$  ( Whole loop with 1 clock pulse )

$$Sum_t = (Sum_{t+1} \oplus \mathcal{S} \cdot Carry_t \oplus A_{it} \otimes B_{0j} \oplus Y_{0j} \otimes N_{0t}) \bmod r$$

$$Carry_t = (Quotient((Sum_{t+1} \oplus Carry_t \oplus A_{it} \otimes B_{0j} \oplus Y_{0j} \otimes N_{0t}), r))$$

(Output of multiplier device in this stage is '0' s)

Second phase:

*Main part*

$$Carry_a = 0$$

For  $j = k$  to  $n - 1$

For  $t = 0$  to  $k - 1$  ( Whole loop with 1 clock pulse )

$$Sum_t = (Sum_{t+1} \oplus \mathcal{S} \cdot Carry_t \oplus A_{it} \otimes B_j \oplus Y_{0t} \otimes N_j) \bmod r$$

$$Carry_t = Quotient((Sum_{t+1} \oplus Carry_t \oplus A_{it} \otimes B_j \oplus Y_{0t} \otimes N_j), r)$$

$$S_{i,j-k} = (S_{i,j-2k} \oplus Sum_0 \oplus \mathcal{S} \cdot Carry_a) \bmod r$$

$$Carry_a = Quotient((S_{i,j-2k} \oplus Sum_0 \oplus Carry_a), r)$$

*Flushing of the multiplier*

For  $j = n$  to  $(n + k - 1)$

For  $t = 0$  to  $k - 1$  ( Whole loop with 1 clock pulse )

$$Sum_t = (Sum_{t+1} \oplus \mathcal{S} \cdot Carry_t) \bmod r$$

$$Carry_t = Quotient((Sum_{t+1} \oplus Carry_t), r)$$

$$S_{i,j-k} = (S_{i,j-2k} \oplus Sum_0 \oplus \mathcal{S} \cdot Carry_a) \bmod r$$

$$Carry_a = Quotient((S_{i,j-2k} \oplus Sum_0 \oplus Carry_a), r)$$

[0084] For a formal explanation with examples of the particular case where  $l = 1$  in the  $GF(p)$  field, see P1.

[0085] The above describes a microelectronic method and apparatus for performing interleaved finite field modular multiplication of integers  $A$  and  $B$  operative to generate an output stream of  $A$  times  $B$  modulus  $N$  wherein  $n$  is the number of characters in the modulus operand register and is larger than  $k$ , wherein the  $\otimes$  multiplication process is performed in iterations, wherein at each interleaved iteration with operands input into a  $\otimes$  multiplying device, consisting of  $N$ , the modulus,  $B$ , a multiplier, a previously computed partial result,  $S$ , and a  $k$  character string segment of  $A$ , a multiplicand, the segments progressing from the  $A_0$  string segment to the  $A_{m-1}$  string segment, wherein each iterative result is  $\oplus$  summated into a next in turn  $S$ , temporary result, in turn, wherein first emitting characters of iterative results are zeroes, the apparatus comprising:

[0086] Typically, there may be four serial  $l$  bit character registers feeding the multiplying device, first ( $B$ ), second ( $S$ ) and third ( $N$ ) and preferably ( $A$ ), configured to efficiently load the multiplier. For computations on long operands which typically are not accommodated in the MAP's internal registers, the CPU can load operands from its accessible memory.

[0087] Typically, these main memory registers store and output operands, respectively operative to store a multiplier value, a partial result value and a modulus,  $N$ .

[0088] The modular multiplying device operative to  $\oplus$  summate into the  $\oplus$  accumulation device, in turn one or two of a plurality of multiplicand values, in turn, during the phases of the iterative  $\otimes$  multiplication process, and in turn to receive as multipliers, in turn, inputs from a first value  $B$  register, second, from an "on the fly" anticipating value,  $Y_0$ , as a multiplier to force first emitting right-hand zero output characters in each iteration, and third values from the modulus,  $N$ , register.

[0089] The multiplicand parallel registers are operative to receive in turn, values from the  $A$ ,  $B$ , and  $N$  register sources, and in turn, also a multiplicand zero forcing  $Y_0$ , value.

[0090] The zero forcing  $Y_0$  detect device is operative to generate a binary string operative to be a multiplier during the first phase of operation and is operative to be a multiplicand in the second phase of each iterative multiplication.

[0091] The multiplicand values to be switched into the  $\oplus$  accumulation device for the first phase consist can be on of four values, a first zero value, a second value,  $A_i$ , which is a  $k$  character string segment of a multiplicand,  $A$ , and a third value  $N_0$ , being the first emitting  $k$  characters of the modulus,  $N$ . The  $N_0$  value is typically switched in at the start of a multiplication, if there is a fourth preload buffer as in Fig. 6. Then when a  $k$  character slice of  $A$  is input, the  $A_i$  value is serially summated with the  $N_0$  value and stored in the fourth buffer.

[0092] If a computation is typically on a single  $k$  character modulus, then there is no need for the  $S$  register, or for the temporary result value,  $S$ . If the operand is  $2k$  characters or longer, then the manipulations must be iterative, with progressing  $A_i$  slices. For squaring operations slices of  $B$  are typically snared from the  $B$  stream on the fly and preloaded into the  $A_i$  preload buffer.

[0093] At the first iteration of a multiplication procedure, the temporary result is zero.

[0094] Subsequent temporary results from previous iterations, are operative to be  $\oplus$  summated with the value emanating from the  $\oplus$  accumulation device, to generate a partial result for the next in turn iteration;

[0095] The multiplicand values to be input, in turn, into the  $\oplus$  accumulation device for the second phase being, a first zero value, which is a pseudo register value, a second  $A_i$  operand, remaining in place from the first phase, and a third  $Y_0$  value having been anticipated in the first phase operative to continue multiplying the remaining characters of the  $N$  modulus.

[0096] The multiplier values input into the multiplying device in the first phase being a first emitting string,  $B_0$ , being the first emitting string segment of the  $B$  operand, concurrently  $\otimes$  multiplying with the second  $\otimes$  multiplier value consisting of the anticipated  $Y_0$  string which is simultaneously loaded character by character as it is generated into a preload multiplicand buffer for the second phase;



The two multiplier values input into the apparatus during the second phase being the left hand  $n - k$  character values from the  $B$  operand, designated  $\underline{B}$ , and the left hand  $n - k$  characters of the  $N$  modulus, designated  $\underline{N}$ , respectively.

The third phase is a flush out of the device operative to transfer the left hand segment of a result value remaining in the  $\oplus$  accumulation device. This can either be a single clock data dump, or a simple serial unload, driven by zero characters fed in the multiplier inputs.

[0097] If the dump is a parallel dump, some means for comparing to decide if the result demands an additional reduction by the modulus.

[0098] One of the more innovative enhancements in the invention, is the reverse mode multiplication in the  $GF(2^q)$ . Because of the lack of interaction between adder cells in this arithmetic, it is possible to perform multiplication and reduction starting from the MS end of the product, thereby having a product that is the modular reduced answer, without a burdensome parasite, caused by disregarded zeroes which, are tantamount to performing a right shift in conventional Montgomery multiplication.

[0099] The second innovation that allows for automatic zero forcing is an extension of the  $Y_0$  function of patent application P2, which describes a device wherein only one bit was anticipated at a time. There the  $J_{00}$  bit, only, multiplied the single bit XORed values. Both the multiplicative inverse of an odd number and its negative value produce an odd number. This saved implementing a look up table or a random logic circuit to compute the  $J_0$  value for  $\ell = 1$ . Note,  $J_0$  is a different quantity in non-alike number systems. We have shown in this extension how a  $Y_0$  value can be derived, for both relevant number fields.

[0100] The following describes the elements of the circuitry operative to anticipate the  $Y_0$  value using first emitting values of the multiplicand, and present inputs of the  $B$  multiplier, carry out values from the  $\oplus$  accumulation device,  $\oplus$  summation values from the  $\oplus$  accumulation device, the present values from the previously computed partial result, and carry out values from the  $\oplus$  adder which  $\oplus$  summates the result from the  $\oplus$  accumulation device with the previous partial result.

[0101] Stated differently, the six values operative to control the zero forcing function, are:

- i. the  $\ell$  bit  $S_{\text{out}}$  bits of the result of the  $\ell$  bit by  $\ell$  bit mod  $2^\ell$   $\otimes$  multiplication of the right-hand character of the  $A_i$  register times the  $B_d$  character of the  $B$  Stream,  $A_0 \otimes B_d \bmod 2^\ell$ ;
- ii. the first emitting carry out character from the  $\oplus$  accumulation device,  $\mathcal{S}(CO_0)$ ;
- iii. the  $\ell$  bit  $S_{\text{out}}$  character from the second from the right-hand character emitting cell of the  $\oplus$  accumulation device,  $SO_1$ ;
- iv. the next in turn character value from the  $S$  stream,  $S_d$ ;
- v. the  $\ell$  bit carry out character from the  $Z$  output full adder,  $\mathcal{S}(CO_z)$ ;
- vi. the  $\ell$  bit  $J_0$  value, which is the negative multiplicative inverse of the right-hand character in the  $N_0$  modulus multiplicand register;

wherein values,  $A_0 \otimes B_d \bmod 2^\ell$ ,  $\mathcal{S}(CO_0)$ ,  $SO_1$ ,  $S_d$  are  $\oplus$  added character to character together and "on the fly"  $\otimes$  multiplied by the  $J_0$  character to output a valid  $Y_0$  zero-forcing anticipatory character to force an  $\ell$  bit egressing character string of zeroes.

Just as in P1, in order to determine if an output must be modular reduced, a sensor operative to compare the output result to  $N$ , the modulus, the mechanism operative to actuate a second subtractor on the output of the result register, thereby to output a modular reduced value which is limited congruent to the output result value precluding the necessity to allot a second memory storage for a smaller result.

**[0102]** The single  $\oplus$  accumulation device, configured to perform multiplication, and an anticipating zero forcing mechanism together are operative to perform a series of interleaved  $\otimes$  modular multiplications and squarings. The total device performs the equivalent of three integer multiplications, as in a conventional Montgomery method,  $J_0$  is a  $k$  character device multiplying the first  $k$  character summation of  $B_0 \otimes A_i$  and  $S_i$ , and in finally using the  $Y_0$  to multiply  $N$ .

**[0103]** Whilst the SuperMAP is computing the last iteration of a multiplication, the first slice of a next multiplication can be preloaded into a preload register buffer means on the fly. This value may be the result of a previous multiplication or a slice of a multiplicand residing in one of the register segments in the register bank of Fig. 1 or Fig. 5.

[0104] The preloaded value which is a  $\oplus$  summation of two multiplicands is  $\oplus$  summated into a  $k$  character register, only, for  $GF(2^q)$  computations. In  $GF(p)$  computations, provision must be made for an additional carry bit.

[0105] Especially for very long moduli, buffers and registers adjacent to the SuperMAP typically have insufficient memory resources. Means for loading operands directly into preload buffers is provided, operative to store operands in the CPU's memory map. For reverse format multiplication, bit order of input words from the CPU are typically reversed in the Data In and Data Out devices.

### BRIEF DESCRIPTION OF THE DRAWINGS

[0106] In the drawings:

Thick lines designate  $k$  character ( $k/\text{bit}$ ) wide parallel bus lines. Thinner contiguous signal lines depict  $l$  bit wide lines. Most control lines are not depicted; those that are included are typically necessary to understand procedures and are typically drawn as dash-dot-dash lines

Fig. 1 is a block diagram of the apparatus according to an embodiment of the invention where four main registers are depicted and the serial data flow path to the operational unit is shown and the input and output data path to the host CPU of Fig. 3;

Fig. 2 is a block diagram of an embodiment of an operational unit operative to manipulate data from Fig. 1;

Fig. 3 is a simplified block diagram of a preferred embodiment of a complete single chip, monolithic cryptocomputer, typically in smart cards;

Fig. 4 is a simplified block diagram of a preferred embodiment of a complete single chip monolithic cryptocomputer wherein a data disable switch is operative to provide for accelerated unloading of data from the operational unit;

Fig. 5 is a simplified block diagram of a data register bank, operative to generate  $J_0$ ;

Fig. 6 is a simplified block diagram of an operational unit, wherein the  $Y_0$  sense is a device operative to force a zero first phase output;

Fig. 7A is a block diagram of the main computational part of Fig. 6, with circled numbered sequence icons relating to the timing diagrams and flow charts of Figs. 7B, 7C, and Fig. 7D;

Fig. 7B is an event timing pointer diagram showing progressively the process leading to and including the first iteration of a squaring operation;

Fig. 7C is a detailed event sequence to eliminate the "Next Montgomery Squaring" delays in the first iteration of a squaring sequence iconed pointers relating to Fig. 7A, Fig. 7B, and Fig. 7D;

Fig. 7D illustrates the timing of the computational output, relating to Fig. 7A, 7B, and Fig. 7C; and

Figs. 8A and 8B, taken together describe generation of the  $Y_0$  vector in  $GF(2^q)$  and in  $GF(p)$ . Fig. 8A is a set of look up tables for determining the negative multiplicative inverse of the right hand character of  $N_0$ , for  $\ell = 2$  and  $\ell = 4$  and Fig. 8B describes, in simplified block form the signals that generate the  $Y_0$  function for  $\ell = 2$  and  $\ell = 4$  in both number fields.

## DESCRIPTION OF PREFERRED EMBODIMENTS

[0107] In the drawings:

Thick lines designate  $k$  character ( $k\ell$  bit) wide parallel bus lines. Thinner contiguous connecting signal lines depict  $\ell$  bit wide lines. Typically, control lines are not depicted; those that are preferably necessary to understand procedures, are typically drawn as dash-dot-dash lines

[0108] Figs. 1 - 2, taken together, form a simplified block diagram of a serial-parallel arithmetic logic unit (ALU) constructed and operative in accordance with a preferred embodiment of the present invention. The apparatus of Figs. 1 - 2, preferably include the following components:

[0109] Single Multiplexers - Controlled Switching Elements which select one signal or character stream from a multiplicity of inputs of signals and direct it this chosen signal to a single output. Multiplexers are marked M1 to M13, and are intrinsic parts of larger elements.

[0110] The Multiplexer and pre-adder, 390, is an array of  $k\ell + 1$  multiplexers, and chooses which of the four  $k$  or  $k + 1$  character inputs are to be added into the CSA, 410.

[0111] The B (70 and 80),  $S_A$  (130),  $S_B$  (180), and N (200) and (210) are the four main serial main registers in a preferred embodiment. The  $S_A$  is conceptually and

practically redundant, but can considerably accelerate very long number computations, and save volatile memory resources, especially in the case where the length of the modulus is  $2 \cdot k \cdot m$  characters long.

[0112] Serial Adders and Serial Subtractors are logic elements that have two serial character inputs and one serial character output, and summate or perform subtraction on two long strings of characters. Components 90 and 500 are subtractors, 330, and 460 are serial adders. The propagation time from input to output is very small. Serial subtractors 90 and 500 typically reduce  $B^*$  to  $B$  if  $B^*$  is larger than or equal to  $N$  and/or  $S^*$  to  $S$  if  $S^*$  is larger than or equal to  $N$ . Serial Subtractor 480, is used, as part of a comparator component to detect if  $B^*$  will be larger than or equal to  $N$ . Full Adder 330, adds the two character streams which feed the Load Buffer 340, with a value that is equal to the sum of the values in the 290 and 320 Load Buffers.

[0113] Fast Loaders and Unloaders, 10 and 20, and 30 and 40, respectively, are devices to accelerate the data flow from the CPU controller. Typically, these devices eliminate the necessity for other direct memory access components. 20 and 40 are for reversing the data word, as is necessary for reversing the data words for reverse format  $GF(2^q)$  multiplications.

[0114] Data In, 50, is a parallel in serial out device, as the present ALU device is a serial fed systolic processor, and data is fed in, in parallel, and processed in serial.

[0115] Data Out, 60, is a serial in parallel out device, for outputting results from the coprocessor. The quotient generator is that part of Fig. 2, which generates a quotient character at each iteration of the dividing mechanism.

[0116] Flush Signals on Bd, 240; on  $S^*d$ , 250; and on Nd, 260, are made to assure that the last  $k + 1$  characters can flush out the CSA. A second embodiment would reconcile the  $R$  data at the end of the second phase, and would perform a single parallel data dump to flush out the CSA.

[0117] Load Buffers R1, 290; R2, 320; and R3, 340 are serial in parallel out shift registers adapted to receive the three possible more than zero multiplicand combinations.

[0118] Latches L1, 360; L2, 370; and L3, 380; are made to receive the outputs from the load buffers, thereby allowing the load buffers, the temporal enablement to process the next phase of data before this data is preferably latched into L1, L2, and L3. Latch

LO is typically a “virtual” constant all zero input into 390, which typically is not implemented in latched logic.

[0119]  $Y_0$  Sense, 430, is the logic device, which determines the number of times the modulus is accumulated, in order that a  $k$  character string of LS zeros will exit at  $Z$  in  $\otimes$  multiplications.

[0120] One character delay devices 100, 220 and 230 are inserted in the respective data streams to accommodate for computation synchronization between the data preparation devices in Fig. 1, and the data processing devices in Fig. 1.

[0121] The  $k$  character delay shift register 470, synchronizes  $N$  and the subtractor subtracts  $N$  from the result after disregarding the right hand output zero character string for the larger than  $N$  comparison.

[0122] The Carry Save Accumulator is almost identical to a serial/parallel multiplier, excepting for the fact that three different larger than zero values can be summated, instead of the single value as conventionally is latched onto the input of the s/p multiplier. When used in polynomial based computations "all carry dependent" functions are disabled.

[0123] The Insert Last Carry, 440, is used to insert the  $(m \cdot k \cdot l + 1)$ 'th bit of the  $S$  stream, as the  $S$  register is only  $m \cdot k$  characters long.

[0124] The borrow/overflow detect, 490, typically detects if a result is larger than or equal to the modulus (from  $N$ ), or in  $GF(p)$  computations. In polynomial based computations the overflow is detected if the first significant result bit is a one.

[0125] The control mechanism is not depicted, but is preferably understood to be a set of cascaded counting devices with finite state machines for specific functions with switches set for systolic data flow in both  $GF(p)$  and  $GF(2^q)$ .

[0126] For modular multiplication in the prime and composite prime field of numbers, we define  $A$  and  $B$  to be the multiplicand and the multiplier, and  $N$  to be the modulus which is typically larger than  $A$  or  $B$ .  $N$  also denotes the register where the value of the modulus is stored.  $N$ , may, in some instances, be smaller than  $A$ . We define  $A$ ,  $B$ , and  $N$  as  $m \cdot k = n$  character long operands. Each  $k$  character group will be called a segment, the size of the group defined by the size of the multiplying device. Then  $A$ ,  $B$ , and  $N$  are each  $m$  characters long. For ease in following the step by step procedural explanations, assume that  $A$ ,  $B$ , and  $N$  are 512 bits long, ( $n = 512$ ); assume that  $k$  is 64

characters long because of the present cost effective length of such a multiplier, and data manipulation speeds of simple CPUs; and  $m = 8$  is the number of segments in an operand and also the number of iterations in a squaring or multiplying loop with a 512 bit operand. All operands are positive integers. More generally,  $A, B, N, n, k$  and  $m$  may assume any suitable values.

[0127] In non-modular functions, the  $N$  and  $S$  registers can be used for temporary storage of other arithmetic operands.

[0128] We use the symbol,  $\equiv$ , to denote congruence of modular numbers, for example  $16 \equiv 2 \pmod{7}$ , and we say 16 is congruent to 2 modulo 7 as 2 is the remainder when 16 is divided by 7. When we write  $Y \pmod{N} \equiv X \pmod{N}$ ; both  $Y$  and  $X$  may be larger than  $N$ ; however, for positive  $X$  and  $Y$ , the remainders will be identical. Note also that the congruence of a negative integer  $Y$ , is  $Y + u \cdot N$ , where  $N$  is the modulus, and if the congruence of  $Y$  is to be less than  $N$ ,  $u$  will be the smallest integer which will give a positive result.

[0129] We use the symbol,  $\pmod{N}$ , to denote congruence in a more limited sense. During the processes described herein, a value is often either the desired value, or equal to the desired value plus the modulus. For example  $X \pmod{2} \pmod{7}$ .  $X$  can be equal to 2 or 9. We say  $X$  has limited congruence to 2 mod 7. In the polynomial based field, the analog is a monic value, which we say is larger than  $N$ , and is reduced by XORing to the modulus. As in  $GF(2^q)$ , there is no overflow, this Yen value is typically disregarded.

[0130] When we write  $X = A \pmod{N}$ , we define  $X$  as the remainder of  $A$  divided by  $N$ ; e.g.,  $3 = 45 \pmod{7}$ .

[0131] In number theory the modular multiplicative inverse is a basic concept. For example, the modular multiplicative inverse of  $X$  is written as  $X^{-1}$ , which is defined by  $X \cdot X^{-1} \pmod{N} = 1$ . If  $X = 3$ , and  $N = 13$ , then  $X^{-1} = 9$ , i.e., the remainder of  $3 \cdot 9$  divided by 13 is 1.

[0132] The acronyms MS and LS are used to signify most significant and least significant when referencing bits, characters, segments, and full operand values, as is conventional in digital nomenclature.

[0133] Throughout this specification  $N$  designates both the value  $N$ , and the name of the shift register which contains  $N$ . An asterisk superscript on a value, denotes that

the value, as stands, is potentially incomplete or subject to change.  $A$  is the value of the number which is to be exponentiated, and  $n$  is the character length of the  $N$  operand. After initialization when  $A$  is "Montgomery normalized" to  $A^*$  ( $A^* = 2^n \cdot A$  - to be explained later)  $A^*$  and  $N$  are constant values throughout the intermediate step in the exponentiation. During the first iteration, after initialization of an exponentiation,  $B$  is equal to  $A^*$ .  $B$  is also the name of the register wherein the accumulated value, which finally equals the desired result of exponentiation resides.  $S^*$  designates a temporary value, and  $S$ ,  $S_A$  and  $S_B$  designate, also, the register or registers in which all but the single MS bit of  $S$  is stored. ( $S^*$  concatenated with this MS bit is identical to  $S$ .)  $S(i-1)$  denotes the value of  $S$  at the outset of the  $i$ 'th iteration;  $S_0$  denotes the LS segment of an  $S(i)$ 'th value.

[0134] We refer to the process in the  $GF(p)$  field (defined later)  $\mathcal{P}(A \cdot B)N$  as multiplication in the  $P$  field, or sometimes, simply, a multiplication operation.

[0135] As we have used the standard structure of a serial/parallel multiplier as the basis for constructing a double acting serial parallel multiplier, we differentiate between the summing part of the multiplier, which is based on carry save accumulation, (as opposed to a carry look ahead adder, or a ripple adder, the first of which is considerably more complicated and the second very slow), and call it a carry save adder or accumulator, and deal separately with the preloading mechanism and the multiplexer and latches, which allow us to simultaneously multiply  $A$  times  $B$  and  $C$  times  $D$ , while continuously summate both results with a previous result,  $S$ , e.g.,  $A \cdot B + C \cdot D + S$ , converting this accumulator into a more versatile engine. Additional logic is added to this multiplier in order to provide for an anticipated sense operation necessary for modular reduction and serial summation necessary to provide for modular arithmetic and ordinary integer arithmetic on very large numbers.

### Montgomery Modular Multiplication in $GF(p)$

[0136] The following description refers to Montgomery arithmetic in the  $GF(p)$  of numbers. The present device may be used for Montgomery arithmetic on polynomial based numbers in  $GF(2^q)$ , but would be degraded in performance, as computations would be in the  $P$  field, where all executable operands are multiplied by a factor of  $2^n$ .



[0137] In a classic approach for computing a modular multiplication,  $A \cdot B \bmod N$ , the remainder of the product  $A \cdot B$  is computed by a division process. Implementing a conventional division of large operands is more difficult to perform than serial/parallel multiplications.

[0138] Using Montgomery's modular reduction method, division is essentially replaced by multiplications using two precomputed constants. In the procedure demonstrated herein, there is only one precomputed constant, which is a function of the modulus. This constant is, or can be, computed using this ALU device.

[0139] A simplified presentation of the Montgomery process, as is used in this device is now provided, followed by a complete preferred description.

[0140] If we have an odd number (an LS bit one), e.g., 1010001 ( $=81_{10}$ ) we can always transform this odd number to an even number (a single LS bit of zero) by adding to it another fixing, compensating odd number, e.g., 1111 ( $=15_{10}$ ); as  $1111 + 1010001 = 1100000$  ( $96_{10}$ ). In this particular case, we have found a number that produced five LS zeros, because we knew in advance the whole string, 81, and could easily determine a binary number which we could add to 81, and would produce a new binary number that would have as many LS zeros as we might need. This fixing number must have a right hand one, else it has no effect on the progressive LS characters of a result.

[0141] If our process is a clocked serial/parallel carry save process, where it is desired to have a continuous number of LS zeros, and wherein at each clock cycle we only have to fix the next bit, at each clock it is sufficient to add the fix, if the next bit were to be a one or not to add the fix if the anticipated bit were to be a zero. However, in order not to cause interbit overflows (double carries), this fix is preferably summated previously with the multiplicand, to be added into the accumulator when the relevant multiplier bit is one, and the  $Y$  Sense also anticipates a one.

[0142] Now, as in modular arithmetic, we only are interested in the remainder of a value divided by the modulus, we know that we can add the modulus any number of times to a value, and still have a value that would have the same remainder. This means that we can add  $Y \cdot N = \sum y_i \cdot r^i \cdot N$  to any integer, and still have the same remainder;  $Y$  being the number of times we add in the modulus,  $N$ , to produce the required  $k$  right hand zeros. As described, the modulus that we add can only be odd. (Methods exist

wherein even moduli are defined as  $r^i$  times the odd number that results when  $i$  is the number of LS zeros in the even number.)

[0143] Montgomery interleaved reductions typically reduce storage requirements, and the cost effective size of the multiplication devices. This is especially useful when performing public key cryptographic functions where we multiply one large integer, e.g.,  $n = 1024$  bit, by another same length large integer; a process that would ordinarily produce a double length integer.

[0144] We can add in  $N$ s (the modulus) enough times to  $A \cdot B = X$  or  $A \cdot B + S = X$  during the process of multiplication (or squaring) so that we will have a number,  $Z$ , that has  $n$  LS zeros, and, at most,  $n + 1$  MS characters.

[0145] We can continue using such numbers, disregarding the LS  $n$  characters, if we remember that by disregarding these zeros, we have divided the desired result by  $r^n$ .

[0146] When the LS  $n$  characters are disregarded, and we only use the most significant  $n$  (or  $n + 1$ ) characters, then we have effectively multiplied the result by  $r^{-n}$ , the modular inverse of  $r^n$ . If we would subsequently re-multiply this result by  $r^n \bmod N$  (or  $r^n$ ) we would obtain a value congruent to the desired result (having the same remainder) as  $A \cdot B + S \bmod N$ . As is seen, using MM, the result is preferably multiplied by  $r^{2n}$  to overcome the  $r^{-n}$  parasitic factor reintroduced by the MM.

[0147] Example:

$$A \cdot B + S \bmod N = (12 \cdot 11 + 10) \bmod 13 = (1100 \cdot 1011 + 1010)_2 \bmod 1011_2.$$

$$l = 1, r = 2$$

We will add in  $2^i N$  whenever a fix is necessary on one of the  $n$  LS bits.

	B	1011	
×	A	<u>1100</u>	
add S		1010	
add $A(0) \cdot B$		0000	
		----	sum of LS bit = 0 not add N
add $2^0 (N \cdot 0)$		<u>0000</u>	
sum and shift	0101		→ 0 LS bit leaves carry save
adder			

add $A(1) \cdot B$	0000	
	----	sum of LS bit = 0 - add $N$
add $2^1(N-1)$	<u>1101</u>	
sum and shift	1001	→ 0 LS bit leaves CS adder
add $A(2) \cdot B$	1011	
	----	sum LS bit = 0 don't add $N$
add $2^2(N-0)$	<u>0000</u>	
sum and shift	1010	→ 0 LS bit leaves CS adder
add $A(3) \cdot B$	1011	
	----	sum LS bit = 1 add $N$
add $2^3(N-1)$	<u>1101</u>	
sum and shift	10001	→ 0 LS bit leaves CS adder

And the result is  $10001\ 0000_2 \bmod 13 = 17 \cdot 2^4 \bmod 13$ .

As 17 is larger than 13 we subtract 13, and the result is:

$$17 \cdot 2^4 \equiv 4 \cdot 2^4 \bmod 13.$$

formally  $2^{-n}(A \cdot B + S) \bmod N = 9 \ (12 \cdot 11 + 10) \bmod 13 \equiv 4$

**[0148]** In Montgomery arithmetic we utilize only the MS non-zero result, 4, and effectively remember that the real result has been divided by  $2^n$ ;  $n$  zeros having been forced onto the MM result.

**[0149]** We have added in  $(8+2) \cdot 13 = 10 \cdot 13$  which effectively multiplied the result by  $2^4 \bmod 13 \equiv 3$ . In effect, had we used the superfluous zeros, we can say that we have performed,  $A \cdot B + Y \cdot N + S - (12 \cdot 11 + 10 \cdot 13 + 10)$  in one process, which will be described possible on a preferred embodiment.

**[0150]** Check-  $(12 \cdot 11 + 10) \bmod 13 = 12$ ;  $4 \cdot 3 = 12$ .

**[0151]** In summary, the result of a Montgomery Multiplication is the desired result multiplied by  $2^{-n}$ .

**[0152]** To retrieve the previous result back into a desired result using the same multiplication method, we would have to Montgomery Multiply the previous result by  $2^{2n}$ , which we will call  $H$ , as each MM leaves us with a parasitic factor of  $2^{-n}$ .

[0153] The Montgomery Multiply function  $\mathcal{P}(A \cdot B)N$  performs a multiplication modulo  $N$  of the  $A \cdot B$  product into the  $P$  field. (In the above example, where we derived 4). The retrieval from the  $P$  field back into the normal modular field is performed by enacting  $P$  on the result of  $\mathcal{P}(A \cdot B)N$  using the precomputed constant  $H$ . Now, if  $P \equiv \mathcal{P}(A \cdot B)N$ , it follows that  $\mathcal{P}(P \cdot H)N \equiv A \cdot B \pmod{N}$ ; thereby performing a normal modular multiplication in two  $P$  field multiplications.

[0154] Montgomery modular reduction averts a series of multiplication and division operations on operands that are  $n$  and  $2n$  characters long, by performing a series of multiplications, additions, and subtractions on operands that are  $n$  or  $n + 1$  characters long. The entire process yields a result which is smaller than or equal to  $N$ . For given  $A$ ,  $B$  and odd  $N$  there is always a  $Q$ , such that  $A \cdot B + Q \cdot N$  will result in a number whose  $n$  LS characters are zero, or:

$$P \cdot 2^n = A \cdot B + Q \cdot N$$

[0155] This means that we have an expression that is  $2n$  characters long (with a possible one bit overflow), whose  $n$  LS characters are zero.

[0156] Now, for radix  $r = 2^t$ ; let  $I \cdot r^n \equiv 1 \pmod{N}$  ( $I$  exists for all odd  $N$ ). Multiplying both sides of the previous equation by  $I$  yields the following congruences:  
from the left side of the equation:

$$P \cdot I \cdot r^n \equiv P \pmod{N}; \quad (\text{Remember that } I \cdot r^n \equiv 1 \pmod{N})$$

and from the right side:

$$A \cdot B \cdot I + Q \cdot N \cdot I \equiv A \cdot B \cdot I \pmod{N}; \quad (\text{Remember that } Q \cdot N \cdot I \equiv 0 \pmod{N})$$

therefore:

$$P \equiv A \cdot B \cdot I \pmod{N}.$$

This also means that a parasitic factor  $I = r^{-n} \pmod{N}$  is introduced each time a  $P$  field multiplication is performed.

We define the  $P$  operator such that:

$$P \equiv A \cdot B \cdot I \pmod{N} \equiv \mathcal{P}(A \cdot B)N.$$

and we call this "multiplication of  $A$  times  $B$  in the  $\mathcal{P}$  field", or Montgomery Multiplication.

The retrieval from the  $P$  field can be computed by operating  $\mathcal{P}$  on  $P \cdot H$ , making:

$$\mathcal{P}(P \cdot H)N \equiv A \cdot B \bmod N ;$$

We can derive the value of  $H$  by substituting  $P$  in the previous congruence.

We find:

$$\mathcal{P}(P \cdot H)N \equiv (A \cdot B \cdot I)(H)(I) \bmod N ;$$

(see that  $A \cdot B \cdot I \leftarrow P$ ;  $H \leftarrow H$ ;  $I \leftarrow$  and any multiplication operation introduces a parasitic  $I$ )

**[0157]** If  $H$  is congruent to the multiple inverse of  $I^2$  then the congruence is valid, therefore:

$$H = I^{-2} \bmod N \equiv r^{2n} \bmod N$$

( $H$  is a function of  $N$  and we call it the  $H$  parameter)

**[0158]** In conventional Montgomery methods, to enact the  $P$  operator on  $A \cdot B$ , the following process may be employed, using the precomputed constant  $J$ :

- 1)  $X = A \cdot B$
- 2)  $Y = (X \cdot J) \bmod r^n$  (only the  $n$  LS characters are necessary)
- 3)  $Z = X + Y \cdot N$
- 4)  $S = Z / r^n$  (The requirement on  $J$  is that it forces  $Z$  to be divisible by  $r^n$ )
- 5)  $P \nless S \bmod N$  ( $N$  is to be subtracted from  $S$ , if  $S \geq N$ )

Finally, at step 5):

$$P \nless \mathcal{P}(A \cdot B)N,$$

[After the subtraction of  $N$ , if necessary:

$$P = \mathcal{P}(A \cdot B)N ]$$

Following the above:

$$Y = A \cdot B \cdot J \bmod r^n \text{ (using only the } n \text{ LS characters);}$$

and:

$$Z = A \cdot B + (A \cdot B \cdot J \bmod r^n) \cdot N.$$

**[0159]** In order that  $Z$  be divisible by  $r^n$  (the  $n$  LS characters of  $Z$  are preferably zero) and the following congruence will exist:

$$[A \cdot B + (A \cdot B \cdot J \bmod r^n) \cdot N] \bmod r^n \equiv 0$$

**[0160]** In order that this congruence will exist,  $N \cdot J \bmod r^n$  is congruent to -1 or:

$$J \equiv -N^{-1} \bmod r^n.$$

and we have found the constant  $J$ .

[0161]  $J$ , therefore, is a precomputed constant which is a function of  $N$  only. However, in a machine that outputs a MM result, character by character, provision should be made to add in  $N$ s at each instance where the output character in the LS string would otherwise have been a zero, thereby obviating the necessity of precomputing  $J$  and subsequently computing  $Y = A \cdot B \cdot J \bmod r^n$ , as  $Y$  can be detected character by character using hardwired logic. We have also described that this method can only work for odd  $N$ s.

[0162] Therefore, as is apparent, the process described employs three multiplications, one summation, and a maximum of one subtraction, for the given  $A$ ,  $B$ ,  $N$ , and a precomputed constant to obtain  $\mathcal{P}(A \cdot B)N$ . Using this result, the same process and a precomputed constant,  $H$ , (a function of the module  $N$ ) we are able to find  $A \cdot B \bmod N$ . As  $A$  can also be equal to  $B$ , this basic operator can be used as a device to square or multiply in the modular arithmetic.

### Interleaved Montgomery Modular Multiplication

[0163] The previous section describes a method for modular multiplication which involved multiplications of operands which were all  $n$  characters long, and results which required  $2n + 1$  characters of storage space.

[0164] Using Montgomery's interleaved reduction as described in P1, it is possible to perform the multiplication operations with shorter operands, registers, and hardware multipliers; enabling the implementation of an electronic device with relatively few logic gates.

[0165] First we will describe how the device can work, if at each iteration of the interleave, we compute the number of times that  $N$  is added, using the  $J_0$  constant. Later, we describe how to interleave, using a hardware derivation of  $Y_0$ , which will eliminate the  $J_{0+}$  phase of each multiplication  $\{(2) \text{ in the following example}\}$ , and enable us to integrate the functions of two separate serial/multipliers into the new single generic multiplier which can perform  $A \cdot B + C \cdot N + S$  at better than double speed using similar silicon resources.

[0166] Using a  $k$  character multiplier, it is convenient to define segments of  $k$  character length; there are  $m$  segments in  $n$  characters; i.e.,  $m \cdot k = n$ .

$J_0$  will be the LS segment of  $J$ .

[0167] Therefore:

$$J_0 \equiv -N_0^{-1} \bmod r^k \text{ (} J_0 \text{ exists as } N \text{ is odd).}$$

[0168] Note, the  $J$  and  $J_0$  constants are compensating numbers that when enacted on the unreduced output, tell us how many times to add the modulus, in order to have a predefined number of least significant zeros. We will later describe an additional advantage to the present serial device; since, as the next serial bit of output can be easily determined, we can always add the modulus (always odd) to the next intermediate result. This is the case if, without this addition, the output character, the LS serial bit exiting the CSA, would have been a “1”; thereby adding in the modulus to the previous even intermediate result, and thereby promising another LS zero in the output string. Remember, congruency is maintained, as no matter how many times the modulus is added to the result, the remainder is constant.

[0169] In the conventional use of Montgomery's interleaved reduction,  $\mathcal{P}(A \cdot B)N$  is enacted in  $m$  iterations as described in steps (1) to (5):

Initially  $S(0) = 0$  (the  $\forall$  value of  $S$  at the outset of the first iteration).

For  $i = 1, 2 \dots m$ :

1)  $X = S(i-1) + A_{i-1} \cdot B$  ( $A_{i-1}$  is the  $i-1$  'th character of  $A$  ;  $S(i-1)$  is the value of  $S$  at the outset of the  $i$  'th iteration.)

2)  $Y_0 = X_0 \cdot J_0 \bmod r^k$  (The LS  $k$  characters of the product of  $X_0 \cdot J_0$ )  
(The process uses and computes the  $k$  LS characters only, e.g., the least significant 64 characters) In the preferred implementation, this step is obviated, because in a serial machine  $Y_0$  can be anticipated character by character.

$$3) \quad Z = X + Y_0 \cdot N$$

4)  $S(i) = Z/r^k$  (The  $k$  LS characters of  $Z$  are always 0, therefore  $Z$  is always divisible by  $r^k$ . This division is tantamount to a  $k$  character right shift as the LS  $k$  characters of  $Z$  are all zeros; or as will be seen in the circuit, the LS  $k$  characters of  $Z$  are simply disregarded.

5)  $S(i) = S(i) \bmod N$  ( $N$  is to be subtracted from those  $S(i)$ 's which are larger than  $N$ ). Finally, at the last iteration (after the subtraction of  $N$ , when necessary),  $C = S(m) = \mathcal{P}(A \cdot B)N$ .

[0170] To derive  $F = A \cdot B \bmod N$ , the  $P$  field computation,  $\mathcal{P}(C \cdot H)N$ , is performed.

[0171] It is desired to know, in a preferred embodiment, that for all  $S(i)$ 's,  $S(i)$  is smaller than  $2N$ . This also means, that the last result ( $S(m)$ ) can always be reduced to a quantity less than  $N$  with, at most, one subtraction of  $N$ .

[0172] We observe that for operands which are used in the process:

$S(i-1) < r^{n+1}$  (the temporary register can be one bit longer than the  $B$  or  $N$  register),

$$B < N < r^n \text{ and } A_{i-1} < r^k.$$

By definition:

$S(i) = Z/r^k$  (The value of  $S$  at the end of the process, before a possible subtraction )

For all  $Z$ ,  $Z(i) < r^{n+k+1}$ .

$$X_{\max} = S_{\max} + A_i \cdot B < r^{n+1} - 1 + (r^k - 1)(r^n - 1)$$

$$Q_{\max} = Y_0 N < (r^k - 1)(r^n - 1)$$

therefore:

$$Z_{\max} < r^{k+n+1} - r^{k+1} + 1 < r^{k+n+1} - 1.$$

and as  $Z_{\max}$  is divided by  $r^k$ :

$$S(m) < r^{n+1} - r^1.$$

Because  $N_{\min} > r^n - r$ ,  $S(m)_{\max}$  is always less than  $2 \cdot N_{\min}$ , and therefore, one subtraction is all that is necessary on a final result.

$$S(m)_{\max} - N_{\min} = (r^{n+1} - r^1 - 1) - (r^n - 1) = r^n - 4 < N_{\min}.$$

### Example of a Montgomery interleaved modular multiplication:

[0173] The following computations in the hexadecimal format clarify the meaning of the interleaved method:

$N = \text{a59}$ , (the modulo),  $A = \text{99b}$ , (the multiplier),  $B = \text{5c3}$  (the multiplicand),  $n = 12$ ,  $r = 2$ , (the character length of  $N$ ),  $k = 4$ , (the size in characters of the multiplier and also the size of a segment), and  $m = 3$ , as  $n = k \cdot m$ .



$J_0 = 7$  as  $7 \cdot 9 \equiv -1 \pmod{16}$  and  $H \equiv 2^2 \cdot 12 \pmod{a59} \equiv 44b$ .

[0174] The expected result is  $F \equiv A \cdot B \pmod{N} \equiv 99b \cdot 5c3 \pmod{a59} \equiv 375811 \pmod{a59} = 220_{16}$ .

[0175] Initially:  $S(0) = 0$

*Step 1*  $X = S(0) + A_0 \cdot B = 0 + b \cdot 5c3 = 3f61$

$$Y_0 = X_0 \cdot J_0 \pmod{r^k} = 7 \text{ (} Y_0 \text{ - hardwire anticipated in SuperMAP)}$$

$$Z = X + Y_0 \cdot N = 3f61 + 7 \cdot a59 = 87d0$$

$$S(1) = Z / r^k = 87d$$

*Step 2*  $X = S(1) + A_1 \cdot B = 87d + 9 \cdot 5c3 = 3c58$

$$Y_0 = X_0 \cdot J_0 \pmod{r^k} = 8 \cdot 7 \pmod{2^4} = 8 \text{ (Hardwire anticipated)}$$

$$Z = X + Y_0 \cdot N = 3c58 + 52c8 = 8f20$$

$$S(2) = Z / r^k = 8f2$$

*Step 3*  $X = S(2) + A_2 \cdot B = 8f2 + 9 \cdot 5c3 = 3ccd$

$$Y_0 = d \cdot 7 \pmod{2^4} = b \text{ (Hardwire anticipated)}$$

$$Z = X + Y_0 \cdot N = 3ccd + b \cdot a59 = aea0$$

$$S(3) = Z / r^k = aea,$$

as  $S(3) > N$ ,

$$S(m) = S(3) - N = aea - a59 = 91$$

Therefore  $C = \mathcal{P}(A \cdot B)N = 91_{16}$ .

Retrieval from the  $P$  field is performed by computing  $\mathcal{P}(C \cdot H)N$ :

Again initially:  $S(0) = 0$

*Step 1*  $X = S(0) + C_0 \cdot H = 0 + 1 \cdot 44b = 44b$

$$Y_0 = d \text{ (Hardwire anticipated in SuperMAP)}$$

$$Z = X + Y_0 \cdot N = 44b + 8685 = 8ad0$$

$$S(1) = Z / r^k = 8ad$$

*Step 2*  $X = S(1) + C_1 \cdot H = 8ad + 9 \cdot 44b = 2f50$

$$Y_0 = 0 \text{ (Hardwire anticipated in SuperMAP)}$$

$$Z = X + Y_0 \cdot N = 2f50 + 0 = 2f50$$

$$S(2) = Z / r^k = 2f5$$

$$\text{Step 3} \quad X = S(2) + C_2 \cdot H = 2f5 + 0 \cdot 44b = 2f5$$

$$Y_0 = 3 \text{ (Hardwire anticipated in SuperMAP)}$$

$$Z = X + Y_0 \cdot N = 2f5 + 3 \cdot a59 = 2200$$

$$S(3) = Z / r^k = 220_{16}$$

which is the expected value of  $99b \cdot 5c3 \bmod a59$ .

[0176] If at each step we disregard  $k$  LS zeros, we are in essence multiplying the  $n$  MS characters by  $r^k$ . Likewise, at each step, the  $i$ 'th segment of the multiplier is also a number multiplied by  $r^{ik}$ , giving it the same rank as  $S(i)$ .

[0177] It can also be noted that in another preferred embodiment, wherein it is of some potential value to know the  $J_0$  constant, if  $A_i \cdot B + S = 1$ ; then  $Y_0 = -N_0^{-1} = J_0$

Exponentiation:

[0178] The following derivation of a sequence [D. Knuth, *The art of computer programming*, vol. 2: Seminumerical algorithms, Addison-Wesley, Reading Mass., 1981] hereinafter referred to as "Knuth", explains a sequence of squares and multiplies, which implements a modular exponentiation.

[0179] After precomputing the Montgomery constant,  $H = 2^{2^n}$ , as this device can both square and multiply in the  $P$  field, we compute:

$$C = A^E \bmod N.$$

Let  $E(j)$  denote the  $j$  bit in the binary representation of the exponent  $E$ , starting with the MS bit whose index is 1 and concluding with the LS bit whose index is  $q$ , we can exponentiate as follows for odd exponents:

$$A^* \nleftarrow \mathcal{P}(A \cdot H)N \quad A^* \text{ is now equal to } A \cdot 2^n.$$

$$B = A^*$$

FOR  $j = 2$  TO  $q-1$

$$B \nleftarrow \mathcal{P}(B \cdot B)N$$

IF  $E(j) = 1$  THEN

$$B \nleftarrow \mathcal{P}(B \cdot A^*)N$$

ENDFOR

$B \nexists \mathcal{P}(B \cdot A)N \quad E(0) = 1; B$  is the last desired temporary result multiplied by  $2^n$ ,

$A$  is the original  $A$ .

$$C = B$$

$$C = C - N \text{ if } C \geq N.$$

After the last iteration, the value  $B$  is  $\nexists$  to  $A^E \bmod N$ , and  $C$  is the final value.

[0180] To clarify, we shall use the following example:

$$E = 1011 \longrightarrow E(1) = 1; E(2) = 0; E(3) = 1; E(4) = 1;$$

To find  $A^{1011} \bmod N; q = 4$

$$A^* = \mathcal{P}(A \cdot H)N = A \cdot I^{-2} I = A \cdot I^{-1} \bmod N$$

$$B = A^*$$

FOR  $j = 2$  to  $q$

$$B = \mathcal{P}(B \cdot B)N \text{ which produces: } A^{2(I-1)2 \cdot I} = A^{2 \cdot I-1}$$

$$E(2) = 0; \quad B = A^{2 \cdot I-1}$$

$$j = 3 \quad B = \mathcal{P}(B \cdot B)N = A^{2(I-1)2 \cdot I} = A^{4 \cdot I-1}$$

$$E(3) = 1 \quad B = \mathcal{P}(B \cdot A^*)N = (A^{4 \cdot I-1}) (A \cdot I^{-1}) \cdot I = A^{5 \cdot I-1}$$

$$j = 4 \quad B = \mathcal{P}(B \cdot B)N = A^{10 \cdot I-2 \cdot I} = A^{10 \cdot I-1}$$

As  $E(4)$  was odd, the last multiplication will be by  $A$ , to remove the parasitic  $I^{-1}$ .

$$B = \mathcal{P}(B \cdot A)N = A^{10 \cdot I-1 \cdot A \cdot I} = A^{11}$$

$$C = B$$

[0181] A method for computing the  $H$  parameter by a reciprocal process is described in US Patent 5,513,133.

[0182] Reference is now made to Fig. 3, which is a simplified block diagram showing how the present invention may be implemented in smart cards and other security devices. An internal bus, 500, links components including a CPU, 502, a RAM, 504, non-volatile memory, 506, controlled access EEPROM, 508, and modular arithmetic coprocessor, 510. As shown herein, the coprocessor, 510, is connected via data, 512, and control, 514, registers to the internal bus, 500. The controlled access ROM, 508, is connected via address and data latch means, 516, and a control and test

register, 518. Various other devices may be attached to the bus such as a physical sequence random generator, 520, security logic, 522, smart card and external port interfacing circuitry, 524, and 526, respectively.

[0183] When a cryptographic program, such as verifying an RSA signature is executed, it may require modular arithmetic functions such as modular exponentiation. The cryptographic program that calls the cryptographic function is preferably run on the CPU, 502.

[0184] Reference is now made to Fig. 4, which is another simplified block diagram of an implementation of the present invention for use in a smart card. Parts that are the same as those shown in Fig. 3 are given the same reference numerals and are not described again, except as necessary for an understanding of the present embodiment. In Fig. 4 the CPU 502 is shown with an external accumulator 7350. Data Disable Switch, 7340, detaches the CPU Accumulator from the Data Bus 500, while unloading data from the arithmetic coprocessor enables direct transfer of data from the SMAP to memory.

[0185] Fig. 5 is a simplified block diagram of a preferred embodiment of a data register bank, 6205, within a coprocessor 6075, as depicted in coprocessors of Figs. 2, 6 and 7, with a  $J_0$  generator, wherein the  $J_0$  generator typically compiles an  $\ell$  bit primary zero forcing function.

[0186] The coprocessor 6075 is connected to a data bus with a CPU as in previous figures. A register bank, 6205, comprises a  $B$  register 6070, an  $A$  register 6130, an  $S$  register 6180, and an  $N$  register 6200. The outputs of each of the registers are connected to a serial data switch and serial process conditioner 6020, which in turn is connected to an operational unit, 6206, which carries out the modular arithmetic operations. Connected between the  $N$  register, 6200, and the operational unit, 6206, is a  $J_0$  generator, 552.

[0187] In the embodiment the  $J_0$  generator compiles an  $\ell$  bit primary zero forcing function for use in the modular arithmetic functions described above.

[0188] Fig. 6 is a simplified internal block diagram of the operational unit of Fig. 5. The unit, preferably supports accelerated squaring operations, in that the additional  $Y_0B_0$  serial buffer accepts  $Y_0$  in the first phase, and in the second phase a modular reduced  $B_0$  for a subsequent squaring operation, wherein it is found that  $B$  is larger than  $N$ .

[0189] Reference is now made to Fig. 7A, which is a block diagram of the main computational part of the operational unit of Fig. 6. Numbers appearing in circles relate to the sequence diagrams of Figs. 7B and 7D.

[0190] Reference is now made to Fig. 7B, which is an event timer pointer diagram showing progressively the process leading to and including the first iteration of a squaring operation.

[0191] Reference is now made to Fig. 7C which is a generalized event sequence showing a method of eliminating the Next Montgomery Squaring delays in a first iteration of a squaring sequence. Circled numbers refer to Figs. 7A, 7B and 7D.

[0192] Reference is now made to Fig. 7D which is a generalized event timer pointer diagram illustrating the timing of the computational output of the first iteration of a squaring operation.

[0193] Reference is now made to Fig 8A, a set of look up tables, which typically show the choices of  $J_0$ , which is the negative of the multiplicative inverse over modulus  $2^l$  of the right hand character of  $N_0$ . As  $N_0$  is always either monic for  $GF(2^q)$  or odd for  $GF(p)$ ,  $J_0$  always exists.

[0194] In Figs. 8A and 8B, we refer to this right hand character of the modulus as  $N_0$ . We refer to  $N_{0j}$  as the  $j$ 'th bit of the locally defined  $N_0$  character

[0195] Fig. 8B is a schematic for designing either a 4 bit or a 2 bit  $Y_0$  zero forcing function character. The variable inputs into the force function are the  $N_0$  bits (constant throughout a multiplication), the  $l$ ,  $S_0$  bits, and the  $l$  right hand bits of the product of the  $l$  multiplier and multiplicand bits,  $A_{i0}$  and  $B_{0j}$ , and the carry switch,  $\mathcal{S}$ , which determines whether functions work in  $GF(2^q)$  or  $GF(p)$ . The  $A$  and  $B$  bits are input into a  $\otimes$  multiplier and  $\oplus$  added to the  $S_0$ . When  $\mathcal{S} = 0$ , all carries are disabled.

[0196] It is appreciated that various features of the invention, which are, for clarity, described in the contexts of separate embodiments, may also be provided in combination in a single embodiment. Conversely, various features of the invention, which are, for brevity, described in the context of a single embodiment, may also be provided separately or in any suitable subcombination.

[0197] It will be appreciated by persons skilled in the art, that the present invention is not limited to what has been particularly shown and described hereinabove. Rather,

the scope of the present invention includes both combinations and subcombinations of the various features described hereinabove as well as variations and modifications thereof, which would occur to persons skilled in the art upon reading the foregoing description which are not in the prior art.

[0198] In the following claims, symbols such as have the meanings given in the preceding description.